

RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRRRRRRRRRRR		MMM		MMM	SSSSSSSSSSSS
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMMMMM	MMMMMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRR	RRR	MMM	MMM	SSS	
RRRRRRRRRRRR		MMM	MMM	SSSSSSSSSS	
RRRRRRRRRRRR		MMM	MMM	SSSSSSSSSS	
RRRRRRRRRRRR		MMM	MMM	SSSSSSSSSS	
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM		SSS
RRR	RRR	MMM	MMM	SSSSSSSSSSSS	
RRR	RRR	MMM	MMM	SSSSSSSSSSSS	
RRR	RRR	MMM	MMM	SSSSSSSSSSSS	

SY

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

NT

PI

```
RRRRRRRR  MM      MM      333333  SSSSSSSS  SSSSSSSS  IIIIII  DDDDDDDD  RRRRRRRR
RRRRRRRR  MM      MM      333333  SSSSSSSS  SSSSSSSS  IIIIII  DDDDDDDD  RRRRRRRR
RR      RR  MMMM  MMMM  33      33  SS      SS      SS      II      DD      DD  RR      RR
RR      RR  MMMM  MMMM  33      33  SS      SS      SS      II      DD      DD  RR      RR
RR      RR  MM  MM  MM      33      33  SS      SSSSSS  SSSSSS  II      DD      DD  RR      RR
RRRRRRRR  MM      MM      33      33  SSSSSS  SSSSSS  II      DD      DD  RRRRRRRR
RRRRRRRR  MM      MM      33      33  SSSSSS  SSSSSS  II      DD      DD  RRRRRRRR
RR      RR  MM      MM      33      33  SS      SS      SS      II      DD      DD  RR      RR
RR      RR  MM      MM      33      33  SS      SS      SS      II      DD      DD  RR      RR
RR      RR  MM      MM      33      33  SS      SS      SS      II      DD      DD  RR      RR
RR      RR  MM      MM      33      33  SSSSSSSS  SSSSSSSS  IIIIII  DDDDDDDD  RR      RR
RR      RR  MM      MM      333333  SSSSSSSS  SSSSSSSS  IIIIII  DDDDDDDD  RR      RR
RR      RR  MM      MM      333333  SSSSSSSS  SSSSSSSS  IIIIII  DDDDDDDD  RR      RR
```

```
LL      IIIIII  SSSSSSSS
LL      IIIIII  SSSSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SSSSSS
LL      II      SSSSSS
LL      II      SS
LL      II      SS
LL      II      SS
LL      II      SS
LLLLLLLLLL  IIIIII  SSSSSSSS
LLLLLLLLLL  IIIIII  SSSSSSSS
```

.....

```
1 0001 0 MODULE RM3SSIDR (LANGUAGE (BLISS32) ,
2 0002 0 IDENT = 'V04-000'
3 0003 0 ) =
4 0004 1 BEGIN
5 0005 1
6 0006 1 *****
7 0007 1 *
8 0008 1 * COPYRIGHT (c) 1978, 1980, 1982, 1984 BY
9 0009 1 * DIGITAL EQUIPMENT CORPORATION, MAYNARD, MASSACHUSETTS.
10 0010 1 * ALL RIGHTS RESERVED.
11 0011 1 *
12 0012 1 * THIS SOFTWARE IS FURNISHED UNDER A LICENSE AND MAY BE USED AND COPIED
13 0013 1 * ONLY IN ACCORDANCE WITH THE TERMS OF SUCH LICENSE AND WITH THE
14 0014 1 * INCLUSION OF THE ABOVE COPYRIGHT NOTICE. THIS SOFTWARE OR ANY OTHER
15 0015 1 * COPIES THEREOF MAY NOT BE PROVIDED OR OTHERWISE MADE AVAILABLE TO ANY
16 0016 1 * OTHER PERSON. NO TITLE TO AND OWNERSHIP OF THE SOFTWARE IS HEREBY
17 0017 1 * TRANSFERRED.
18 0018 1 *
19 0019 1 * THE INFORMATION IN THIS SOFTWARE IS SUBJECT TO CHANGE WITHOUT NOTICE
20 0020 1 * AND SHOULD NOT BE CONSTRUED AS A COMMITMENT BY DIGITAL EQUIPMENT
21 0021 1 * CORPORATION.
22 0022 1 *
23 0023 1 * DIGITAL ASSUMES NO RESPONSIBILITY FOR THE USE OR RELIABILITY OF ITS
24 0024 1 * SOFTWARE ON EQUIPMENT WHICH IS NOT SUPPLIED BY DIGITAL.
25 0025 1 *
26 0026 1 *
27 0027 1 *****
28 0028 1
29 0029 1 ++
30 0030 1
31 0031 1 FACILITY: RMS32 INDEX SEQUENTIAL FILE ORGANIZATION
32 0032 1
33 0033 1 ABSTRACT:
34 0034 1 Search thru a SISR array and return the first
35 0035 1 non-deleted record
36 0036 1
37 0037 1
38 0038 1 ENVIRONMENT:
39 0039 1
40 0040 1 VAX/VMS OPERATING SYSTEM
41 0041 1
42 0042 1 --
43 0043 1
44 0044 1
45 0045 1 AUTHOR: Todd M. Katz RE-CREATION DATE: 23-Jun-1982
46 0046 1
47 0047 1
48 0048 1 MODIFIED BY:
49 0049 1
50 0050 1 V03-022 RAS0164 Ron Schaefer 29-Jun-1983
51 0051 1 Make references to RMSRU_RECLAIM be LONG_RELATIVE addressing.
52 0052 1
53 0053 1 V03-021 MCN0002 Maria del C. Nasr 24-Mar-1983
54 0054 1 More linkages reorganization
55 0055 1
56 0056 1 V03-020 TMK0013 Todd M. Katz 11-Mar-1983
57 0057 1 Make a change to RMSFOLLOW_PTR so that the primary data bucket
```

58	0058	1	containing the target record is exclusively accessed if it is
59	0059	1	possible that some reclamation maybe done (the file is write
60	0060	1	accessed and Recovery Unit Journallable).
61	0061	1	
62	0062	1	V03-019 MCN0001 Maria del C. Nasr 28-Feb-1983
63	0063	1	Reorganize linkages
64	0064	1	
65	0065	1	V03-018 TMK0012 Todd M. Katz 18-Jan-1983
66	0066	1	Add support for Recovery Unit Journalling and RU ROLLBACK
67	0067	1	Recovery of ISAM files. Support involves modifications to
68	0068	1	the routine RMSFOLLOW_PTR.
69	0069	1	
70	0070	1	The purpose of the routines within this module is to find the
71	0071	1	next non-deleted primary data record by means of an alternate
72	0072	1	index. Towards this goal, RMS will search all SIDs with key
73	0073	1	values matching the key in keybuffer 2 according to the
74	0074	1	characteristics of the search, until such a record is found. If
75	0075	1	during its search RMS encounters records that are marked
76	0076	1	RU_DELETE, RMS will try and delete them for good at this time
77	0077	1	provided it has write access to the file and the Recovery Unit
78	0078	1	in which they were deleted has completed successfully.
79	0079	1	
80	0080	1	If RMS is able to delete a primary data record marked RU_DELETE,
81	0081	1	then RMS proceeds to continue looking for a non-deleted primary
82	0082	1	data record just as if it had encountered a deleted record in
83	0083	1	the first place. Likewise, if RMS is unable to delete a record
84	0084	1	that is marked RU_DELETE because it does not have write access
85	0085	1	to the file, it will also continue its search. However, if RMS
86	0086	1	is unable to delete a RU_DELETE marked record for good because
87	0087	1	the Recovery Unit in which it was marked RU_DELETE has not
88	0088	1	successfully terminated, then RMS returns this record as if it
89	0089	1	was the next non-deleted primary data record, and lets a higher
90	0090	1	higher level routine decide whether or not to wait for the
91	0091	1	Recovery Unit in which the record was deleted to complete, or to
92	0092	1	return an error to the user.
93	0093	1	
94	0094	1	RMS will also re-format any records that are marked RU_UPDATE
95	0095	1	and are in a special format provided the stream has write access
96	0096	1	to the file, and the Recovery Unit in which the record was
97	0097	1	updated has terminated.
98	0098	1	
99	0099	1	In addition to this change, I have made a further enhancement to
100	0100	1	the routine RMSSEARCH_SIDR. At the present time, when a SIDR
101	0101	1	array element is encountered that should have been marked
102	0102	1	deleted, it is marked deleted without recovering any space if
103	0103	1	and only if the stream has write access to the file, and the
104	0104	1	file is not being shared. I have changed this by removing the
105	0105	1	restriction that to mark a SIDR element deleted the file can not
106	0106	1	be shared. Furthermore, RMS will now not only mark such elements
107	0107	1	as deleted, but reclaim as much space as possible as long as the
108	0108	1	stream has write access to the file.
109	0109	1	
110	0110	1	V03-017 TMK0011 Todd M. Katz 05-Jan-1983
111	0111	1	Eliminate the routine RMSFND_SDR_ARRAY. This global routine is
112	0112	1	now being called in only one place, and has been folded directly
113	0113	1	into the code.
114	0114	1	

115 0115 1
116 0116 1
117 0117 1
118 0118 1
119 0119 1
120 0120 1
121 0121 1
122 0122 1
123 0123 1
124 0124 1
125 0125 1
126 0126 1
127 0127 1
128 0128 1
129 0129 1
130 0130 1
131 0131 1
132 0132 1
133 0133 1
134 0134 1
135 0135 1
136 0136 1
137 0137 1
138 0138 1
139 0139 1
140 0140 1
141 0141 1
142 0142 1
143 0143 1
144 0144 1
145 0145 1
146 0146 1
147 0147 1
148 0148 1
149 0149 1
150 0150 1
151 0151 1
152 0152 1
153 0153 1
154 0154 1
155 0155 1
156 0156 1
157 0157 1
158 0158 1
159 0159 1
160 0160 1
161 0161 1
162 0162 1
163 0163 1
164 0164 1
165 0165 1
166 0166 1
167 0167 1
168 0168 1
169 0169 1
170 0170 1
171 0171 1

- V03-016 TMK0010 Todd M. Katz 04-Jan-1983
If during positioning to the next primary data record by means of an alternate index, RMS encounters a SIDR element that is not marked deleted although the primary data record it points to is either deleted or the alternate key does not match the key of the SIDR, then RMS may mark the element deleted provided the file is not being shared and has been open for write access before it continues its search for the next primary data record. If either of these two conditions is violated, RMS can not mark the element deleted, but just positions past it to continue the search. What it should also be doing when it positions past the element in this case is increment the count of the number of preceeding elements in the current SIDR array. RMS was not doing this and this represents a bug.
- V03-015 TMK0009 Todd M. Katz 07-Dec-1982
Remove all the re-positioning code. Sigh. This code was originally required to fix the 1.5 SIDR deadlock problem. I have come up with a different way to fix this problem. I have changed how buckets are locked during \$DELETEs. A stream can now never hold onto a primary data bucket while attempting to access a SIDR bucket. Therefore, a stream is able to hold onto a SIDR bucket while waiting for a primary data bucket without the possibility of a deadlock existing. Thus, there is no longer any need to perform any type of SIDR re-positioning, and I have eliminated all the new code that used to do it.
- V03-014 TMK0008 Todd M. Katz 12-Nov-1982
The routine RMS\$FND_SDR_ARRAY requires as input, the size of the secondary key it is to search for in IR\$B_KEYSZ. This IRAB cell might not have the correct key size when this routine is called from within RM\$SIDR_REPOS. Yet, RM\$SIDR_REPOS can not merely overwrite the cell with the correct key size because RMS might be performing a generic search at the time SIDR re-positioning was required, and might later need the value stored there. Therefore, RMS must save the value in IR\$B_KEYSZ when the routine RM\$SIDR_REPOS is entered, set it to the full key size, and restore the original value when the routine is exited. If this is not done, the possibility exists that RMS\$FND_SDR_ARRAY might position RMS to the wrong SIDR instead of to the SIDR it was positioned to when the routine was first entered.
- V03-013 TMK0007 Todd M. Katz 09-Nov-1982
Make another change to the SIDR re-positioning code. IF RMS is unable to position either to the primary data record, or back to the SIDR during SIDR re-positioning, then before returning RMS ascertains whether it was unsuccessful in positioning to the current SIDR array, or whether it was unsuccessful at positioning even to the SIDR itself. The former case represents an error, the latter is reasonable because the SIDR could have been deleted and the space it occupied reclaimed. During this attempt to re-position to the current SIDR, performed by the routine RMS\$FIND_SIDR, RMS will position either to the current SIDR (which would represent an error), or to the next SIDR that followed this SIDR. However, if RMS is unable to position to the current SIDR and there is no next SIDR (the end-of-file is

172 0172 1
173 0173 1
174 0174 1
175 0175 1
176 0176 1
177 0177 1
178 0178 1
179 0179 1
180 0180 1
181 0181 1
182 0182 1
183 0183 1
184 0184 1
185 0185 1
186 0186 1
187 0187 1
188 0188 1
189 0189 1
190 0190 1
191 0191 1
192 0192 1
193 0193 1
194 0194 1
195 0195 1
196 0196 1
197 0197 1
198 0198 1
199 0199 1
200 0200 1
201 0201 1
202 0202 1
203 0203 1
204 0204 1
205 0205 1
206 0206 1
207 0207 1
208 0208 1
209 0209 1
210 0210 1
211 0211 1
212 0212 1
213 0213 1
214 0214 1
215 0215 1
216 0216 1
217 0217 1
218 0218 1
219 0219 1
220 0220 1
221 0221 1
222 0222 1
223 0223 1
224 0224 1
225 0225 1
226 0226 1
227 0227 1
228 0228 1

encountered), then RMS\$FIND_SIDR returns an error of RNF. Unfortunately, the routine RM\$SIDR_REPOS can not return this status to its caller (RM\$SEARCH_SIDR) because that would indicate that RMS was unable to find the primary data record during re-positioning - a very grave error and definitely not the case here. When RM\$SIDR_REPOS returns an error of RNF signalling this very bad condition, RMS converts it into a BUG error, and returns it to the user. However, the error RMS does want to return from RM\$SEARCH_SIDR in the case when RMS finds both the primary data record and the SIDR deleted during re-positioning and is unable to position to the next SIDR because of encountering the end-of-file is in fact RNF! In order to arrange things so that this error will be returned under these circumstances, I have changed RM\$SIDR_REPOS so that it will return an error of EOF when RM\$FIND_SIDR returns RNF and RM\$SEARCH_SIDR so that it changes the EOF returned by RM\$SIDR_REPOS back to RNF.

V03-012 TMK0006 Todd M. Katz 29-Oct-1982
If RMS is successful at finding the target primary data record within RMS\$FOLLOW_PTR, then save the current index descriptor and replace it with the index descriptor for the primary key before determining record overhead and size of the primary data record. After this determination is made, the descriptor is restored.

V03-011 TMK0005 Todd M. Katz 22-Sep-1982
If a SIDR re-positioning is required, and key compression is enabled, then reset the variable containing the address of the last record in the bucket encountered with a zero front compressed key to be the first record in the bucket. This resetting is only required when RMS must continue the search for a non-deleted primary data record within the current SIDR, and when the possibility exists that the search maybe continued in a SIDR with a completely different key value necessitating its expansion into keybuffer 2. This resetting is necessary because during the re-positioning RMS had to release the SIDR bucket, and then reclaim it, and while it didn't have it locked, anything might have happened to its contents. The reason why the variable is initialized with the address of the first SIDR in the bucket is because the key of the first SIDR must be zero front compressed, and it is the last known record with such a property.

V03-010 KBT0329 Keith B. Thompson 22-Sep-1982
Change check for sharing to sfsb test

V03-009 TMK0004 Todd M. Katz 06-Sep-1982
The field IRB\$B_SRCHFLAGS is now a word. Fix all references to it.

The routine RM\$SEARCH_SIDR maybe called to search SIDR arrays for a non-deleted primary data record either when RMS is positioning sequentially, or when RMS is positioning randomly by an alternate key value. When I initially wrote this routine I mistakenly wrote it for the sequential case only. Since in this case RM\$SEARCH_SIDR only has to search those SIDRs whose keys exactly match the full size search key, this mean't that

229 0229 1
230 0230 1
231 0231 1
232 0232 1
233 0233 1
234 0234 1
235 0235 1
236 0236 1
237 0237 1
238 0238 1
239 0239 1
240 0240 1
241 0241 1
242 0242 1
243 0243 1
244 0244 1
245 0245 1
246 0246 1
247 0247 1
248 0248 1
249 0249 1
250 0250 1
251 0251 1
252 0252 1
253 0253 1
254 0254 1
255 0255 1
256 0256 1
257 0257 1
258 0258 1
259 0259 1
260 0260 1
261 0261 1
262 0262 1
263 0263 1
264 0264 1
265 0265 1
266 0266 1
267 0267 1
268 0268 1
269 0269 1
270 0270 1
271 0271 1
272 0272 1
273 0273 1
274 0274 1
275 0275 1
276 0276 1
277 0277 1
278 0278 1
279 0279 1
280 0280 1
281 0281 1
282 0282 1
283 0283 1
284 0284 1
285 0285 1

positioning randomly by key was broken for all cases involving generic search keys, greater-than searches, or greater-than or equal searches when more than one SIDR had to be searched. The fix for this set of problems was actually quite simple. If the search characteristics are not setup such that RMS is performing an exact match search, the routine RM\$SEARCH_SIDR saves the search key in keybuffer 5, and extracts the key of the current SIDR into keybuffer 2. When this SIDR is exhausted and a non-deleted primary data record has not been found, RMS returns the search key saved in keybuffer 5 to keybuffer 2, and determines whether the key of the next SIDR matches the search key according to the characteristics of the search. If so, this whole cycle repeats itself, otherwise, this routine returns the appropriate error to its caller.

I also made a change to RM\$SIDR_REPOS involving the case when RMS is unable to position to a primary data record because it has been deleted, and when it re-positions back to the SIDR it finds that it too has been deleted. In such a case, because RMS was unable to re-position to the SIDR, it did a greater-than or equal search and had positioned exactly to the SIDR that followed. This is exactly where RMS wants to continue its search for a non-deleted primary data record provided the key of this SIDR matches search key according to the search characteristics. Therefore, all the routine RM\$SIDR_REPOS has to do is return a 0. The routine RM\$SEARCH_SIDR, when it sees this 0, knows that the current SIDR has been exhausted (or in this case deleted), and that it should go determine whether the search can continue with the SIDR it now finds itself positioned to.

V03-008 KBT0298 Keith B. Thompson 24-Aug-1982
Reorganize psects

V03-007 TMK0003 Todd M. Katz 10-Aug-1982
Change the linkage of RM\$SIDR_REPOS. The address of the beginning of the SIDR is now both in the input parameter list and in the output parameter list. This is necessary instead of just passing its address to this routine, an address of a stack location, because this routine will allow RMS to stall, and when RMS resumes after a stall, the stack addresses are not necessarily the same as they were before the stall.

V03-006 TMK0002 Todd M. Katz 10-Aug-1982
When RMS positions to the first non-deleted primary data record by alternate key, it first skips past all entries within a SIDR that are marked deleted until either the end of the SIDR is encountered, or a non-deleted entry is found. Using this non-deleted entry, RMS attempts to position to the primary data record. If RMS finds that the data record is in fact deleted, it wants to mark the SIDR entry deleted, and return to the loop that looks for a non-deleted SIDR entry. There it starts its search for a non-deleted SIDR array element with the current SIDR element which of course RMS has just marked deleted. However, RMS can only mark such a SIDR entry deleted if it has write access to the file and the file is not being shared. Thus, the way this RMS currently works, if the

```

286 0286 1  file was being shared or had not been opened for write access
287 0287 1  and RMS has positioned to a non-deleted S IDR array element
288 0288 1  which pointed to a deleted primary data record, RMS would not
289 0289 1  mark the S IDR entry deleted, it would return to the loop which
290 0290 1  searches for the first non-deleted S IDR entry starting with the
291 0291 1  current entry, and it would position to the very same
292 0292 1  non-deleted entry (which points to a deleted primary data
293 0293 1  record). The result is an infinite loop! To fix this problem,
294 0294 1  RMS will now immediately position past the current element to
295 0295 1  the next one when it finds that the current non-deleted S IDR
296 0296 1  element points to a deleted primary data record, and it is
297 0297 1  unable to mark the S IDR element deleted at that point.
298 0298 1
299 0299 1  V03-005 TMK0001      Todd M. Katz      22-Jun-1982
300 0300 1  Revised entire module.
301 0301 1
302 0302 1  *****
303 0303 1
304 0304 1  LIBRARY 'RMSLIB:RMS';
305 0305 1
306 0306 1  REQUIRE 'RMSSRC:RMSIDXDEF';
307 0371 1
308 0372 1  ! Define default PSECTS for code
309 0373 1  !
310 0374 1  PSECT
311 0375 1      CODE = RMSRMS3(PSECT_ATTR),
312 0376 1      PLIT = RMSRMS3(PSECT_ATTR);
313 0377 1
314 0378 1  ! Linkages
315 0379 1  !
316 0380 1  LINKAGE
317 0381 1      L_PRESERVE1,
318 0382 1      L_RABREG_56?,
319 0383 1      L_RABREG_67,
320 0384 1      L_RABREG_7,
321 0385 1      L_REC_OVHD,
322 0386 1      L_SIDR_FIRST,
323 0387 1      L_JSBOT,
324 0388 1
325 0389 1  ! Linkages for Local Routines
326 0390 1  !
327 0391 1      RLS$FOLLOW_PTR = JSB ( )
328 0392 1      : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN),
329 0393 1      RLS$POS_BY_COUNT = JSB (REGISTER = 1)
330 0394 1      : GLOBAL (COMMON_RABREG, R_REC_ADDR, R_IDX_DFN);
331 0395 1
332 0396 1  ! External Routines
333 0397 1  !
334 0398 1  EXTERNAL ROUTINE
335 0399 1      RMS$COMPARE_REC      : RLSRABREG_67,
336 0400 1      RMS$CSEARCH_TREE     : RLSRABREG_67,
337 0401 1      RMS$EXT_ARRAY_RFA   : RLSRABREG_67,
338 0402 1      RMS$FIND_BY_RRV     : RLSRABREG_67,
339 0403 1      RMS$KEY_DEST        : RLSRABREG_7,
340 0404 1      RMS$GETNXT_ARRAY    : RLSRABREG_67,
341 0405 1      RMS$RECORD_KEY      : RLS$PRESERVE1,
342 0406 1      RMS$REC_OVHD       : RLS$REC_OVHD,
```

RM3SSIDR
V04-000

J 9
16-Sep-1984 02:07:52
14-Sep-1984 13:01:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[RMS.SRC]RM3SSIDR.B32;1 Page 7 (1)

```
: 343      0407 1      RMSRLSBKT      : RL$PRESERVE1,
: 344      0408 1      RMSRU RECLAIM  : RL$RABREG_67 ADDRESSING_MODE(LONG_RELATIVE),
: 345      0409 1      RM$SIDR_END    : RL$RABREG_67,
: 346      0410 1      RM$SIDR_FIRST  : RL$SIDR_FIRST,
: 347      0411 1      RM$SQUISH_SIDR : RL$RABREG_567,
: 348      0412 1      RM$UNPACK_REC  : RL$JSB01;
: 349      0413 1
: 350      0414 1      ! Forward Routine
: 351      0415 1      !
: 352      0416 1      FORWARD ROUTINE
: 353      0417 1      RM$POS_BY_COUNT : RL$POS_BY_COUNT;
```

```

355 0418 1 %SBTTL 'RMSFOLLOW_PTR'
356 0419 1 ROUTINE RMSFOLLOW_PTR (VBN, ID) : RL$FOLLOW_PTR =
357 0420 1
358 0421 1 ++
359 0422 1
360 0423 1 FUNCTIONAL DESCRIPTION:
361 0424 1
362 0425 1 This routine recieves as input the RFA of an allegedly non-deleted
363 0426 1 primary data record containing a secondary key of specific value. It
364 0427 1 positions to that primary data record, and checks whether it is deleted
365 0428 1 and if not, whether the secondary key whose specific value is stored
366 0429 1 in keybuffer 2 has been deleted from the primary data record.
367 0430 1
368 0431 1 If RMS finds that the target primary data record is marked RU DELETE
369 0432 1 and the Recovery Unit in which the record was deleted is still active,
370 0433 1 then RMS returns positioned to this record and lets a higher level
371 0434 1 routine decide what to do. If the Recovery Unit in which the record was
372 0435 1 deleted has successfully terminated, then RMS will return an error of
373 0436 1 DEL after deleting this RU DELETED record (if it has write access to
374 0437 1 the file), and releasing the primary data bucket.
375 0438 1
376 0439 1 If RMS encounters a record that is marked RU UPDATE and is in a special
377 0440 1 format then RMS will return positioned to this record after
378 0441 1 reformatting it. The reformatting is done if RMS has write access to
379 0442 1 the file, and the Recovery Unit in which it was updated has
380 0443 1 successfully terminated.
381 0444 1
382 0445 1 CALLING SEQUENCE:
383 0446 1
384 0447 1 BSBW RMSFOLLOW_PTR ()
385 0448 1
386 0449 1 INPUT PARAMETERS:
387 0450 1
388 0451 1 VBN - RFA VBN of the target primary data record
389 0452 1 ID - RFA ID of the target primary data record
390 0453 1
391 0454 1 IMPLICIT INPUTS:
392 0455 1
393 0456 1 IDX_DFN - address of current index descriptor
394 0457 1 -IDX$B_KEYSZ - size of secondary key
395 0458 1 IDX$W_MINRECSZ - minimum size of record to contain key
396 0459 1
397 0460 1 IFAB - address of IFAB
398 0461 1 IFB$B_KBUFSZ - size of each keybuffer
399 0462 1 IFB$B_PLG_VER - prologue version of file
400 0463 1 IFB$V_RU - if set, the file is Recovery Unit Journallable
401 0464 1 IFB$V_WRTACC - if set, file is open for write access
402 0465 1
403 0466 1 IRAB - address of IRAB
404 0467 1 IRB$L_CURBDB - address of SDR bucket's BDB
405 0468 1 (0 if re-positioning)
406 0469 1 IRB$L_KEYBUF - address of keybuffers
407 0470 1 IRB$L_RECBUF - address of internal record buffer
408 0471 1
409 0472 1 REC_ADDR - address of SDR array element
410 0473 1 (invalid if re-positioning)
411 0474 1
```

```

: 412 0475 1 OUTPUT PARAMETERS:
: 413 0476 1 NONE
: 414 0477 1
: 415 0478 1 IMPLICIT OUTPUTS:
: 416 0479 1
: 417 0480 1 IRAB
: 418 0481 1 IRB$L_CURBDB - address of primary data bucket's BDB (success)
: 419 0482 1 or contents on input (failure)
: 420 0483 1 IRB$L_NXTBDB - contents on input (success)
: 421 0484 1 or 0 (failure)
: 422 0485 1 IRB$L_RECBUF - unpacked primary data record
: 423 0486 1 (success and prologue 3 file)
: 424 0487 1 IRB$V_RU_DELETE - if set, do not reclaim SDR array space
: 425 0488 1
: 426 0489 1 REC_ADDR - address of primary data bucket (success)
: 427 0490 1 or contents on input (failure)
: 428 0491 1
: 429 0492 1 ROUTINE VALUE:
: 430 0493 1
: 431 0494 1 SUC - non-deleted primary data record successfully positioned to.
: 432 0495 1 DEL - primary data record deleted or specific alternate key deleted
: 433 0496 1 from primary data record.
: 434 0497 1 RNF - primary data record not found.
: 435 0498 1 various I/O errors
: 436 0499 1
: 437 0500 1 SIDE EFFECTS:
: 438 0501 1
: 439 0502 1 On success, REC_ADDR points to the primary data record (which will be
: 440 0503 1 unpacked if the file's prologue version is 3), the primary data
: 441 0504 1 bucket's BDB address is stored in IRB$L_CURBDB, and IRB$L_NXTBDB
: 442 0505 1 contains whatever was in IRB$L_CURBDB on input.
: 443 0506 1 On any and all failures, REC_ADDR points to whatever it had pointed to
: 444 0507 1 on input, IRB$L_CURBDB contains whatever it contained on input,
: 445 0508 1 and any buckets accessed have been released.
: 446 0509 1 If the record is marked RU_DELETE, it might have been deleted.
: 447 0510 1 If the record is marked RU_UPDATE or RU_DELETE and RMS is unable to
: 448 0511 1 recover any space from it because the Recovery Unit in which the
: 449 0512 1 record was modified has not completed, then RMS sets the state
: 450 0513 1 bit IRB$V_RU_DELETE whenever it makes a decision to return an error
: 451 0514 1 status of RMS$DEL and the file has been opened for write access.
: 452 0515 1 This will guarantee that the corresponding SDR array element will
: 453 0516 1 only be marked RU_DELETE, and no space will be reclaimed from it.
: 454 0517 1 If the record is marked RU_UPDATED, it might have been reformatted.
: 455 0518 1
: 456 0519 1 --
: 457 0520 1
: 458 0521 2 BEGIN
: 459 0522 2
: 460 0523 2 BUILTIN
: 461 0524 2 AP;
: 462 0525 2
: 463 0526 2 EXTERNAL REGISTER
: 464 0527 2 COMMON_RAB_STR,
: 465 0528 2 R_IDX_DFN_STR,
: 466 0529 2 R_REC_ADDR_STR;
: 467 0530 2
: 468 0531 2 LOCAL
```

```

: 469 0532 2      SAVE_SDR_ADDR,
: 470 0533 2      STATUS;
: 471 0534 2
: 472 0535 2      ! Save the current SDR array's address, and the address of the SDR
: 473 0536 2      ! bucket's BDB while performing the primary data record lookup.
: 474 0537 2
: 475 0538 2      SAVE_SDR_ADDR = .REC_ADDR;
: 476 0539 2      IRAB[IRB$L_NXTBDB] = .IRAB[IRB$L_CURBDB];
: 477 0540 2
: 478 0541 2      ! If the file is write accessed and Recovery Unit Journallable, then make
: 479 0542 2      ! sure the primary data bucket containing the target record is exclusively
: 480 0543 2      ! accessed in case reclamation is required.
: 481 0544 2
: 482 0545 2      IF .IFAB[IFB$V_WRTACC]
: 483 0546 2          AND
: 484 0547 2          .IFAB[IFB$V_RU]
: 485 0548 2      THEN
: 486 0549 2          IRAB[IRB$B_CACHEFLGS] = CSH$M_LOCK;
: 487 0550 2
: 488 0551 2      ! Position to the primary data record pointing at by the SDR array element
: 489 0552 2      ! RMS has positioned to.
: 490 0553 2
: 491 0554 2      STATUS = RMSFIND_BY_RRV (.VBN, .ID, 0);
: 492 0555 2
: 493 0556 2      ! If RMS is successful at positioning to the target primary data record
: 494 0557 2      ! only to find that it has been deleted within a Recovery Unit, subject
: 495 0558 2      ! this record to further processing before deciding what to do with this
: 496 0559 2      ! record.
: 497 0560 2
: 498 0561 2      IF .STATUS
: 499 0562 2          AND
: 500 0563 2          .REC_ADDR[IRC$V_RU_DELETE]
: 501 0564 2      THEN
: 502 0565 2          BEGIN
: 503 0566 2              LOCAL
: 504 0567 2              TEMP_STATUS;
: 505 0568 2
: 506 0569 2              ! If the Recovery Unit in which the current record was deleted has
: 507 0570 2              ! not completed, or if the file has not been opened for write access
: 508 0571 2              ! then RMS will not be able to reclaim any space from the record. RMS
: 509 0572 2              ! may still want to consider this record as a non-deleted primary data
: 510 0573 2              ! record though.
: 511 0574 2
: 512 0575 2              IF NOT (TEMP_STATUS = RMSRU_RECLAIM())
: 513 0576 2              THEN
: 514 0577 2                  BEGIN
: 515 0578 2
: 516 0579 2                      ! If the current record was deleted within a currently active
: 517 0580 2                      ! Recovery Unit and it was this stream that deleted it; or, if
: 518 0581 2                      ! this stream is able to lock the record but does not have write
: 519 0582 2                      ! access to the file, then RMS wants to treat this record as if it
: 520 0583 2                      ! is actually deleted. To do so, RMS will return a status of
: 521 0584 2                      ! RMS$DEL so that the search will continue. However, RMS will also
: 522 0585 2                      ! set the state bit IRB$V_RU_DELETE, if it has write access to the
: 523 0586 2                      ! file, so that no space is reclaimed from the corresponding SDR
: 524 0587 2                      ! array element.
: 525 0588 2
```

```

526      0589  4      !
527      0590  5      IF .TEMP_STATUS<0,16> NEQU RMSERR(RLK)
528      0591  4      THEN
529      0592  5          BEGIN
530      0593  5              STATUS = RMSERR(DEL);
531      0594  5          IF .IFAB[IFB$V_WRTACC]
532      0595  5              THEN
533      0596  5                  IRAB[IRB$V_RU_DELETE] = 1;
534      0597  5          END;
535      0598  4      END;
536      0599  4
537      0600  4
538      0601  4      ! The Recovery Unit in which the current record was deleted has
539      0602  4      ! completed successfully and RMS was able to reclaim some space
540      0603  4      ! from the record. RMS changes the status to RMSS_DEL so that the
541      0604  4      ! search will be continued.
542      0605  4
543      0606  3      ELSE
544      0607  3          STATUS = RMSERR(DEL);
545      0608  3
546      0609  3      ! If RMS has discovered that the Recovery Unit in which the target
547      0610  3      ! primary data record was deleted has terminated successfully and that
548      0611  3      ! the record is in fact deleted, then release the primary data bucket
549      0612  3      ! containing the record.
550      0613  3
551      0614  4      IF .STATUS<0,16> EQLU RMSERR(DEL)
552      0615  3      THEN
553      0616  4          BEGIN
554      0617  4              GLOBAL REGISTER
555      0618  4              R_BDB_STR;
556      0619  4              RELEASE (IRAB[IRB$L_CURBDB]);
557      0620  4          END;
558      0621  4      END;
559      0622  3      END;
560      0623  2
561      0624  2
562      0625  2      ! If RMS was successful at positioning to the target primary data record
563      0626  2      ! only to find that it had been updated but not deleted within a Recovery
564      0627  2      ! Unit, and if the file has been opened for write access, then RMS will
565      0628  2      ! try to re-format the record provided the Recovery Unit in which the record
566      0629  2      ! was modified has completed.
567      0630  2
568      0631  2      IF .STATUS
569      0632  2          AND
570      0633  2          .IFAB[IFB$V_WRTACC]
571      0634  2          AND
572      0635  2          .REC_ADDR[IRC$V_RU_UPDATE]
573      0636  2          AND
574      0637  2          NOT .REC_ADDR[IRC$V_RU_DELETE]
575      0638  2      THEN
576      0639  2          RMSRU_RECLAIM();
577      0640  2
578      0641  2      ! If RMS was successful at positioning to the primary data record pointed
579      0642  2      ! at by the SIDR array element, and the record is NOT marked deleted, then
580      0643  2      ! RMS must make sure that it is the right one - ie that the alternate key in
581      0644  2      ! this index matches the key of the SIDR which has been saved in
582      0645  2      ! keybuffer 2.
```

```

: 583      0646 2      !
: 584      0647 2      ! IF .STATUS
: 585      0648 2      ! THEN
: 586      0649 2      ! BEGIN
: 587      0650 2      !
: 588      0651 2      ! LOCAL
: 589      0652 2      !   RECORD_OVHD,
: 590      0653 2      !   RECORD_SIZE,
: 591      0654 2      !   SAVE_UDR_ADDR : REF BBLOCK;
: 592      0655 2      !
: 593      0656 2      ! SAVE_UDR_ADDR = .REC_ADDR;
: 594      0657 2      !
: 595      0658 2      ! ! Determine the size of the primary data record's overhead, and the
: 596      0659 2      ! ! size of the primary data record itself. This is necessary to
: 597      0660 2      ! ! determine whether or not the alternate key has been deleted through
: 598      0661 2      ! ! truncation of part of the record.
: 599      0662 2      !
: 600      0663 4      ! BEGIN
: 601      0664 4      !
: 602      0665 4      ! LOCAL
: 603      0666 4      !   REC_SIZE,
: 604      0667 4      !   SAVE_IDX_DFN : REF BBLOCK;
: 605      0668 4      !
: 606      0669 4      ! SAVE_IDX_DFN = .IDX_DFN;
: 607      0670 4      ! RMSKEY_DESC (0);
: 608      0671 4      !
: 609      0672 4      ! RECORD_OVHD = RMSREC_OVHD(0; REC_SIZE);
: 610      0673 4      !
: 611      0674 4      ! ! If the target primary data record was updated within a Recovery
: 612      0675 4      ! ! Unit then the true size of the primary data record maybe retrieved
: 613      0676 4      ! ! from the last two bytes in the reserved space of the record.
: 614      0677 4      !
: 615      0678 4      ! IF .REC_ADDR[IRC$V_RU_UPDATE]
: 616      0679 4      ! THEN
: 617      0680 5      !   RECORD_SIZE = .(.REC_ADDR + .RECORD_OVHD
: 618      0681 5      !   + .REC_SIZE
: 619      0682 4      !   - IRC$C_DATSZFLD)<0,16>
: 620      0683 4      ! ELSE
: 621      0684 4      !   RECORD_SIZE = .REC_SIZE;
: 622      0685 4      !
: 623      0686 4      ! IDX_DFN = .SAVE_IDX_DFN;
: 624      0687 3      ! END;
: 625      0688 3      !
: 626      0689 3      ! ! If this is a prologue 3 file then the record must be unpacked in
: 627      0690 3      ! ! order to obtain the full size of the record and to be able to compare
: 628      0691 3      ! ! the SIDR key with the alternate key in the data record itself. RMS
: 629      0692 3      ! ! also sets REC_ADDR to point to the unpacked record, so that the
: 630      0693 3      ! ! alternate key may easily be extracted from the record for key
: 631      0694 3      ! ! comparison.
: 632      0695 3      !
: 633      0696 3      ! ! NOTE that because RMS always unpacks prologue 3 primary data records
: 634      0697 3      ! ! when positioning by a secondary key for key comparison, it is never
: 635      0698 3      ! ! necessary to unpack the record a second time during a $GET/$FIND.
: 636      0699 3      !
: 637      0700 3      ! IF .IFAB[IFB$B_PLG_VER] GEQU PLG$C_VER_3
: 638      0701 3      ! THEN
: 639      0702 4      !   BEGIN
```

```

: 640      0703  4
: 641      0704  4      BUILTIN
: 642      0705  4      AP;
: 643      0706  4
: 644      0707  4      GLOBAL REGISTER
: 645      0708  4      R_BKT_ADDR;
: 646      0709  4
: 647      0710  4      REC_ADDR = .REC_ADDR + .RECORD_OVHD;
: 648      0711  4
: 649      0712  4      AP = 0;
: 650      0713  4      RECORD_SIZE = RMSUNPACK_REC (.IRAB[IRB$L_RECBUF], .RECORD_SIZE);
: 651      0714  4
: 652      0715  4      REC_ADDR = .IRAB[IRB$L_RECBUF];
: 653      0716  3      END;
: 654      0717  3
: 655      0718  3      ! If the primary data record is of sufficient size to contain the
: 656      0719  3      ! alternate key (ie - the alternate key has not been deleted through
: 657      0720  3      ! truncation of the record during the process of updating the record
: 658      0721  3      ! by some other process), then RMS determines whether or not the
: 659      0722  3      ! alternate key in the record still matches the SIDR it has positioned
: 660      0723  3      ! to. If this is the case then RMS has found the next non-deleted
: 661      0724  3      ! primary data record, and returns success.
: 662      0725  3
: 663      0726  4      IF (.RECORD_SIZE GEQU .IDX_DFN[IDX$W_MINRECSZ])
: 664      0727  3      AND
: 665      0728  4      (NOT RMSCOMPARE_REC (KEYBUF_ADDR(2), .IDX_DFN[IDX$B_KEYSZ], -1))
: 666      0729  3      THEN
: 667      0730  4      BEGIN
: 668      0731  4      REC_ADDR = .SAVE_UDR_ADDR;
: 669      0732  4      RETURN RMSSUC();
: 670      0733  4      END
: 671      0734  4
: 672      0735  4      ! If the secondary key has been deleted totally from the record through
: 673      0736  4      ! truncation of the record during an $UPDATE by another process, or
: 674      0737  4      ! if during the $UPDATE the secondary key changed, and the secondary
: 675      0738  4      ! key in the primary data record no longer matches the key of the SIDR
: 676      0739  4      ! RMS is positioned to, then this is not the next non-deleted primary
: 677      0740  4      ! data record. In this case, RMS releases the primary data bucket, and
: 678      0741  4      ! sets up to return an error status of record deleted, so that the
: 679      0742  4      ! search for the next non-deleted primary data record can continue.
: 680      0743  4
: 681      0744  3      ELSE
: 682      0745  4      BEGIN
: 683      0746  4
: 684      0747  4      GLOBAL REGISTER
: 685      0748  4      R_BDB_STR;
: 686      0749  4
: 687      0750  4      ! If the target primary data record has been modified within a
: 688      0751  4      ! Recovery Unit, and the stream has write access to the file so
: 689      0752  4      ! that some attempt will be made to reclaim the space occupied by
: 690      0753  4      ! SIDR array element, then set up so that when the SIDR array
: 691      0754  4      ! element pointing to this primary data record is deleted, it is
: 692      0755  4      ! only marked RU_DELETE, and no space is reclaimed.
: 693      0756  4
: 694      0757  4      IF .IFAB[IFB$V_WRTACC]
: 695      0758  4      AND
: 696      0759  5      (.SAVE_UDR_ADDR[IRC$V_RU_DELETE]
```

```

697      0760 5
698      0761 5          OR
699      0762 4          .SAVE_UDR_ADDR[IRC$V_RU_UPDATE])
700      0763 4      THEN
701      0764 4          IRAB[IRB$V_RU_DELETE] = 1;
702      0765 4
703      0766 4      RELEASE(IRAB[IRB$L_CURBDB]);
704      0767 3      STATUS = RMSERR(DEC);
705      0768 3      END;
706      0769 2      END;
707      0770 2
708      0771 2      ! If RMS was unable to position to the target primary data record, or if
709      0772 2      ! after positioning to the record it found that the record itself, or the
710      0773 2      ! secondary key it expected to find there had been deleted, then RMS
711      0774 2      ! restores the address of the SDR bucket's BDB to IRB$L_CURBDB, and the
712      0775 2      ! address of the SDR array element it currently has positioned to
713      0776 2      ! to REC_ADDR, and it returns the appropriate status.
714      0777 2
715      0778 2      IRAB[IRB$L_CURBDB] = .IRAB[IRB$L_NXTBDB];
716      0779 2      IRAB[IRB$L_NXTBDB] = 0;
717      0780 2      REC_ADDR = .SAVE_SDR_ADDR;
718      0781 2
719      0782 2      RETURN .STATUS;
720      0783 2
721      0784 1      END;

```

```
! { of routine }
```

```
.TITLE      RM3SSIDR
.IDENT      \V04-000\

.EXTRN      RM$COMPARE_REC, RM$CSEARCH_TREE
.EXTRN      RM$EXT_ARR? RFA
.EXTRN      RM$FIND BY RRV, RM$KEY_DESC
.EXTRN      RM$GETNXT_ARRAY
.EXTRN      RM$RECORD_KEY, RM$REC OVHD
.EXTRN      RM$RLSBKT, RM$RU RECLAIM
.EXTRN      RM$$SIDR END, RM$$SIDR FIRST
.EXTRN      RM$$SQUISH_SIDR, RM$UNPACK_REC

.PSECT      RM$RMS3, NOWRT, GBL, PIC, 2
```

PC	BB	00000	RM\$FOLLOW	PTR:		
	56	DD 00002		PUSHR	#^M<R2,R3,R4,R5>	0419
				PUSHL	REC_ADDR	0538
04	3C	A9 20	A9 D0 00004	MOVL	32(IRAB), 60(IRAB)	0539
	00A0	0A 06	AA E9 00009	BLBC	6(IFAB), 1\$	0545
	40	CA	01 E1 0000D	BBC	#1, 160(IFAB), 1\$	0547
		A9	01 90 00013	MOVB	#1, 64(IRAB)	0549
			7E D4 00017	CLRL	-(SP)	0554
		20	AE DD 00019	PUSHL	ID	
		20	AE DD 0001C	PUSHL	VBN	
			0000G 30 0001F	BSBW	RM\$FIND_BY_RRV	
			0C C0 00022	ADDL2	#12, SP-	
	5E		50 D0 00025	MOVL	R0, STATUS	
	53		53 E9 00028	BLBC	STATUS, 5\$	0561
3A	53		05 E1 0002B	BBC	#5, (REC_ADDR), 4\$	0563
	66		EF 16 0002F	JSB	RM\$RU_RECLAIM	0576
			00000000G			

82AA	16	50	E8	00035	BLBS	TEMP STATUS, 2\$	
8F	50	B1	00038	CMPW	TEMP STATUS, #33450		0590
	14	13	0003D	BEQL	3\$		
53	8262	8F	3C	0003F	MOVZWL	#33378, STATUS	0593
0B	06	AA	E9	00044	BLBC	6(IFAB), 3\$	0595
07	A9	20	88	00048	BISB2	#32, 7(IRAB)	0597
		05	11	0004C	BRB	3\$	0576
53	8262	8F	3C	0004E	MOVZWL	#33378, STATUS	0607
8F		53	B1	00053	CMPW	STATUS, #33378	0614
		0F	12	00058	BNEQ	4\$	
54	20	A9	D0	0005A	MOVL	32(IRAB), BDB	0621
	20	A9	D4	0005E	CLRL	32(IRAB)	
		7E	D4	00061	CLRL	-(SP)	
		0000G	30	00063	BSBW	RM\$RLSBKT	
5E		04	C0	00066	ADDL2	#4, SP	
12		53	E9	00069	BLBC	STATUS, 5\$	0631
0E	06	AA	E9	0006C	BLBC	6(IFAB), 5\$	0633
66		06	E1	00070	BBC	#6, (REC_ADDR), 5\$	0635
66		05	E0	00074	BBS	#5, (REC_ADDR), 5\$	0637
	00000000G	EF	16	00078	JSB	RM\$RU RECLAIM	0639
03		53	E8	0007E	BLBS	STATUS, 6\$	0647
		0094	31	00081	BRW	13\$	
52		56	D0	00084	MOVL	REC_ADDR, SAVE_UDR_ADDR	0656
55		57	D0	00087	MOVL	IDX_DFN, SAVE_IDX_DFN	0669
		7E	D4	0008A	CLRL	-(SP)	0670
		0000G	30	0008C	BSBW	RM\$KEY_DESC	
5E		04	C0	0008F	ADDL2	#4, SP	
		51	D4	00092	CLRL	R1	0672
		0000G	30	00094	BSBW	RM\$REC OVHD	
66		06	E1	00097	BBC	#6, (REC_ADDR), 7\$	0678
56		50	C1	0009B	ADDL3	RECORD_OVHD, REC_ADDR, R4	0680
	FE	A144	9F	0009F	PUSHAB	-2(REC_SIZE)[R4]	
54		9E	3C	000A3	MOVZWL	@(SP)+, RECORD_SIZE	
		03	11	000A6	BRB	8\$	
54		51	D0	000A8	MOVL	REC_SIZE, RECORD_SIZE	0684
57		55	D0	000AB	MOVL	SAVE_IDX_DFN, IDX_DFN	0686
03	00B7	CA	91	000AE	CMPB	183(IFAB), #3	0700
		16	1F	000B3	BLSSU	9\$	
56		50	C0	000B5	ADDL2	RECORD_OVHD, REC_ADDR	0710
		5C	D4	000B8	CLRL	AP	0712
51		54	D0	000BA	MOVL	RECORD_SIZE, R1	0713
50	68	A9	D0	000BD	MOVL	104(IRAB), R0	
		0000G	30	000C1	BSBW	RM\$UNPACK_REC	
54		50	D0	000C4	MOVL	R0, RECORD_SIZE	
56	68	A9	D0	000C7	MOVL	104(IRAB), REC_ADDR	0715
10		00	ED	000CB	CMPZV	#0, #16, 34(IDX_DFN), RECORD_SIZE	0726
		21	1A	000D1	BGTRU	10\$	
7E		01	CE	000D3	MNEGL	#1, -(SP)	0728
7E	20	A7	9A	000D6	MOVZBL	32(IDX_DFN), -(SP)	
50	00B4	CA	3C	000DA	MOVZWL	180(IFAB), R0	
	60	B940	9F	000DF	PUSHAB	@96(IRAB)[R0]	
		0000G	30	000E3	BSBW	RM\$COMPARE_REC	
5E		0C	C0	000E6	ADDL2	#12, SP	
08		50	E8	000E9	BLBS	R0, 10\$	
56		52	D0	000EC	MOVL	SAVE_UDR_ADDR, REC_ADDR	0731
50		01	D0	000EF	MOVL	#1, R0	0732
		32	11	000F2	BRB	14\$	

RM3SSIDR
V04-000

RMSFOLLOW_PTR

F 10
16-Sep-1984 02:07:52
14-Sep-1984 13:01:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[RMS.SRC]RM3SSIDR.B32;1
Page 16
(2)

04	0C	06	AA	E9	000F4	10\$:	BLBC	6(IFAB), 12\$	:	0757
04	62		05	E0	000F8		BBS	#5, (SAVE_UDR_ADDR), 11\$	:	0759
	62		06	E1	000FC		BBC	#6, (SAVE_UDR_ADDR), 12\$	:	0761
	A9	07	20	88	00100	11\$:	BISB2	#32, 7(IRAB)	:	0763
	54		20	A9	D0 00104	12\$:	MOVL	32(IRAB), BDB	:	0765
			20	A9	D4 00108		CLRL	32(IRAB)	:	
			7E	D4	0010B		CLRL	-(SP)	:	
			0000G	30	0010D		BSBW	RMSRLSBKT	:	
	5E		04	C0	00110		ADDL2	#4, SP	:	
	53	20	8262	8F	3C 00113		MOVZWL	#33378, STATUS	:	0766
	A9		3C	A9	D0 00118	13\$:	MOVL	60(IRAB), 32(IRAB)	:	0778
			3C	A9	D4 0011D		CLRL	60(IRAB)	:	0779
	56			6E	D0 00120		MOVL	SAVE_SDR_ADDR, REC_ADDR	:	0780
	50			53	D0 00123		MOVL	STATUS, R0	:	0782
	5E			04	C0 00126	14\$:	ADDL2	#4, SP	:	0784
			3C	BA	00129		POPR	#^M<R2,R3,R4,R5>	:	
				05	0012B		RSB		:	

; Routine Size: 300 bytes, Routine Base: RMSRMS3 + 0000

```

: 723      0785 1 %SBTTL 'RM$POS_BY_COUNT'
: 724      0786 1 ROUTINE RM$POS_BY_COUNT (SKIP_NUMBER) : RL$POS_BY_COUNT =
: 725      0787 1 ++
: 726      0788 1
: 727      0789 1 FUNCTIONAL DESCRIPTION:
: 728      0790 1
: 729      0791 1     This routine's responsibility is to position to a particular S IDR
: 730      0792 1     array element. It does this by skipping SKIP_NUMBER array elements.
: 731      0793 1
: 732      0794 1 CALLING SEQUENCE:
: 733      0795 1
: 734      0796 1     BSBW RM$POS_BY_COUNT()
: 735      0797 1
: 736      0798 1 INPUT PARAMETERS:
: 737      0799 1
: 738      0800 1     SKIP_NUMBER      - number of array elements to skip
: 739      0801 1
: 740      0802 1 IMPLICIT INPUTS:
: 741      0803 1
: 742      0804 1     REC_ADDR          - address of S IDR
: 743      0805 1
: 744      0806 1 OUTPUT PARAMETERS:
: 745      0807 1     NONE
: 746      0808 1
: 747      0809 1 IMPLICIT OUTPUTS:
: 748      0810 1
: 749      0811 1     REC_ADDR          - address of desired array element
: 750      0812 1
: 751      0813 1 ROUTINE VALUE:
: 752      0814 1
: 753      0815 1     SUC      - positioned to desired array element.
: 754      0816 1     BUG      - the number of array elements preceeding the desired
: 755      0817 1     element exceeds the number of elements in the entire array.
: 756      0818 1
: 757      0819 1 SIDE EFFECTS:
: 758      0820 1
: 759      0821 1     on success, REC_ADDR points to the desired array element.
: 760      0822 1 --
: 761      0823 1
: 762      0824 2 BEGIN
: 763      0825 2
: 764      0826 2 EXTERNAL REGISTER
: 765      0827 2     COMMON RAB STR,
: 766      0828 2     R_IDX_DFN STR,
: 767      0829 2     R_REC_ADDR_STR;
: 768      0830 2
: 769      0831 2 LOCAL
: 770      0832 2     END_OF_S IDR;
: 771      0833 2
: 772      0834 2     ! Find the end of the current S IDR.
: 773      0835 2     !
: 774      0836 2     END_OF_S IDR = RM$SIDR_END();
: 775      0837 2
: 776      0838 2     ! Position to the S IDR's first array element
: 777      0839 2     !
: 778      0840 2     REC_ADDR = RM$SIDR_FIRST(0);
: 779      0841 2
```

```
: 780      0842  2      ! Skip the desired number of array elements, or until the end of the SIDR
: 781      0843  2      ! is encountered - which ever comes first.
: 782      0844  2
: 783      0845  2      DECRU LOOP_INDEX FROM .SKIP_NUMBER TO 1 BY 1
: 784      0846  2      DO
: 785      0847  2
: 786      0848  2      ! If the end of the SIDR is encountered before the desired array
: 787      0849  2      ! element is encountered then this represents an invalid internal
: 788      0850  2      ! condition, and report it as such.
: 789      0851  2
: 790      0852  2      IF .REC_ADDR GEQA .END_OF_SIDR
: 791      0853  2      THEN
: 792      0854  2          RETURN RMSERR(BUG)
: 793      0855  2
: 794      0856  2      ! Position past the current array element to the next array element.
: 795      0857  2      ELSE
: 796      0858  2          RM$GETNXT_ARRAY();
: 797      0859  2
: 798      0860  2      ! Having positioned to the desired SIDR array element, return success.
: 799      0861  2
: 800      0862  2      RETURN RMSSUC();
: 801      0863  2
: 802      0864  1      END;
```

! { of routine }

	1C	BB	00000	RM\$POS_BY_COUNT:		
				POSHR	#*M<R2,R3,R4>	0786
53	51	D0	00002	MOVL	R1, R3	
	0000G	30	00005	BSBW	RM\$SIDR_END	0836
54	50	D0	00008	MOVL	R0, END_OF_SIDR	
	7E	D4	0000B	CLRL	-(SP)	0840
	0000G	30	0000D	BSBW	RM\$SIDR_FIRST	
5E	04	C0	00010	ADDL2	#4, SP	
56	50	D0	00013	MOVL	R0, REC_ADDR	
52	53	D0	00016	MOVL	SKIP_NUMBER, LOOP_INDEX	0852
	11	11	00019	BRB	3\$	
54	56	D1	0001B	1\$: CMPL	REC_ADDR, END_OF_SIDR	
	07	1F	0001E	BLSSU	2\$	
50	8434	8F	3C 00020	MOVZWL	#33844, R0	0854
	0A	11	00025	BRB	4\$	
	0000G	30	00027	2\$: BSBW	RM\$GETNXT_ARRAY	0859
	52	D7	0002A	DECL	LOOP_INDEX	0852
	ED	12	0002C	3\$: BNEQ	1\$	
50	01	D0	0002E	MOVL	#1, R0	0863
	1C	BA	00031	4\$: POPR	#*M<R2,R3,R4>	0864
	05	00033		RSB		

; Routine Size: 52 bytes, Routine Base: RM\$RMS3 + 012C

```

: 804 0865 1 %SBTTL 'RMSSEARCH_SIDR'
: 805 0866 1 GLOBAL ROUTINE RMSSEARCH_SIDR : RL$RABREG_67 =
: 806 0867 1
: 807 0868 1 ++
: 808 0869 1
: 809 0870 1 FUNCTIONAL DESCRIPTION:
: 810 0871 1
: 811 0872 1 The purpose of this routine is to search for a non-deleted primary
: 812 0873 1 data record by means of an alternate key of reference. This routine
: 813 0874 1 starts its search with an array element in the current SIDR array. If
: 814 0875 1 RMS is positioning sequentially, then the array element RMS starts
: 815 0876 1 its search with will always be one more than the value in
: 816 0877 1 IRBSW_SAVE_POS. Thus, RMS may start its search with any of the elements
: 817 0878 1 in the current SIDR array depending upon the value in this variable.
: 818 0879 1 However, if RMS is positioning randomly by key value, then RMS will
: 819 0880 1 always start its search for a non-deleted primary data record with the
: 820 0881 1 very first element in the current SIDR.
: 821 0882 1
: 822 0883 1 When the current SIDR array is exhausted, RMS will attempt to continue
: 823 0884 1 its search within the very next SIDR array that follows. RMS may
: 824 0885 1 continue its search provided the key of this next SIDR matches the
: 825 0886 1 characteristics of the search. These search characteristics include the
: 826 0887 1 search key itself (in keybuffer 2), the size of the search key (in
: 827 0888 1 IRBSB_KEYSZ) which may not be the same as the size of the alternate key
: 828 0889 1 itself, and whether the search is for an equal match, a greater-than
: 829 0890 1 or equal match (IRBSV_SRCHGE is set), or a greater-than match
: 830 0891 1 (IRBSV_SRCHGT is set). If RMS is positioning randomly then any possible
: 831 0892 1 combination of these search characteristics is possible, but when RMS is
: 832 0893 1 positioning sequentially, RMS will only search those SIDRs whose key
: 833 0894 1 exactly matches the full size search key in keybuffer 2.
: 834 0895 1
: 835 0896 1 If during its search for a non-deleted primary data record RMS
: 836 0897 1 encounters a non-deleted SIDR array element pointing to a deleted
: 837 0898 1 primary data record, then RMS will mark the SIDR array element deleted
: 838 0899 1 and reclaim as much space from the SIDR as it can according to the
: 839 0900 1 normal rules of SIDR deletion. However, such activity will only take
: 840 0901 1 place if the stream has write access to the file.
: 841 0902 1
: 842 0903 1 This routine concludes its search when either an error occurs, it is
: 843 0904 1 successful at positioning to a non-deleted primary data record, or it
: 844 0905 1 exhausts all SIDRs matching the search characteristics without
: 845 0906 1 positioning to such a primary data record.
: 846 0907 1
: 847 0908 1 CALLING SEQUENCE:
: 848 0909 1
: 849 0910 1 BSBW RMSSEARCH_SIDR()
: 850 0911 1
: 851 0912 1 INPUT PARAMETERS:
: 852 0913 1 NONE
: 853 0914 1
: 854 0915 1 IMPLICIT INPUTS:
: 855 0916 1
: 856 0917 1 IDX_DFN - address of index descriptor describing key
: 857 0918 1 IDX$V_KEY_COMPR - if set, SIDR key compression is enabled
: 858 0919 1 IDX$B_KEYSZ - size of key
: 859 0920 1
: 860 0921 1 IFAB - address of IFAB
```

```

: 861 0922 1 | IFBSL_SFSB_PTR - pointer to file shared file sync. block
: 862 0923 1 | IFBSW_KBUFSZ - size of a keybuffer
: 863 0924 1 | IFBSV_WRTACC - if set, file is write accessed
: 864 0925 1 |
: 865 0926 1 | IRAB - address of IRAB
: 866 0927 1 | IRBSL_CURBDB - address of SIDER bucket's BDB
: 867 0928 1 | IRBSL_KEYBUF - pointer to the contiguous keybuffers
: 868 0929 1 | IRBSB_KEYSZ - size of search key
: 869 0930 1 | IRBSL_LST_NCMP - address of last zero front compressed key
: 870 0931 1 | IRBSV_PRM - if set, permanence should be set in SIDER BDB
: 871 0932 1 | IRBSW_SAVE_POS - number of elements preceeding starting point
: 872 0933 1 | IRBSV_SRCHGE - if set, search for GE match
: 873 0934 1 | IRBSV_SRCHGT - if set, search for GT match
: 874 0935 1 |
: 875 0936 1 | REC_ADDR - address of SIDER within bucket
: 876 0937 1 |
: 877 0938 1 | OUTPUT PARAMETERS:
: 878 0939 1 | NONE
: 879 0940 1 |
: 880 0941 1 | IMPLICIT OUTPUTS:
: 881 0942 1 |
: 882 0943 1 | IRAB - address of IRAB
: 883 0944 1 | IRBSL_CURBDB - address of primary data bucket's BDB
: 884 0945 1 | IRBSW_FIRST_ID - RFA ID of new current SIDER's first element
: 885 0946 1 | IRBSL_FIRST_VBN - RFA VBN of new current SIDER's first element
: 886 0947 1 | IRBSL_LST_NCMP - address of last zero front compressed key
: 887 0948 1 | IRBSL_RFA_VBN - VBN of new current SIDER's bucket
: 888 0949 1 | IRBSW_SAVE_POS - number of elements before new current element
: 889 0950 1 |
: 890 0951 1 | REC_ADDR - address of primary data record
: 891 0952 1 |
: 892 0953 1 | ROUTINE VALUE:
: 893 0954 1 |
: 894 0955 1 | BUG - invalid internal condition
: 895 0956 1 | SUC - next non-deleted primary data record found
: 896 0957 1 | RNF - all SIDERs of this key value exhausted
: 897 0958 1 | various I/O errors
: 898 0959 1 |
: 899 0960 1 | SIDE EFFECTS:
: 900 0961 1 |
: 901 0962 1 | AP is trashed.
: 902 0963 1 | Keybuffer 5 will contain the search key, if this routine was called
: 903 0964 1 | with a generic search key (search key size less than key size),
: 904 0965 1 | or to find the first SIDER with a key greater-than or
: 905 0966 1 | greater-than-or-equal to the search key in keybuffer 2.
: 906 0967 1 | On success, IRBSL_CURBDB contains the primary data bucket's BDB
: 907 0968 1 | address, the SIDER bucket will have been released, REC_ADDR
: 908 0969 1 | points to the primary data bucket, and the IRAB contains
: 909 0970 1 | sufficient information necessary to uniquely identify and
: 910 0971 1 | re-position to the SIDER array element pointing to the
: 911 0972 1 | non-deleted primary data record. In addition, keybuffer 2 will
: 912 0973 1 | contain the key of the SIDER.
: 913 0974 1 | On all errors except for a RNF found error, the SIDER bucket will have
: 914 0975 1 | been released and possibly marked dirty/permanent.
: 915 0976 1 | SIDER array elements and in fact whole SIDERs might have been deleted
: 916 0977 1 | if the stream positioning has write access to the file.
: 917 0978 1 |
```

```

918 0979 1 !--
919 0980 1
920 0981 2 BEGIN
921 0982 2
922 0983 2 BUILTIN
923 0984 2 AP,
924 0985 2 TESTBITSC;
925 0986 2
926 0987 2 EXTERNAL REGISTER
927 0988 2 COMMON RAB_STR,
928 0989 2 R_IDX_DFN_STR,
929 0990 2 R_REC_ADDR_STR;
930 0991 2
931 0992 2 LABEL
932 0993 2 NEXT;
933 0994 2
934 0995 2 LOCAL
935 0996 2 BEG_OF_SIDR,
936 0997 2 END_OF_SIDR,
937 0998 2 FLAGS : BLOCK [1],
938 0999 2 STATUS;
939 1000 2
940 1001 2 MACRO
941 1002 2 FIRST_TIME = 0,0,1,0 %,
942 1003 2 NEW_KEY = 0,1,1,0 %;
943 1004 2
944 1005 2 FLAGS[FIRST_TIME] = 1;
945 1006 2
946 1007 2 ! Save the address of the SIDR, and the address of the first byte past the
947 1008 2 ! last SIDR array element (ie the end of the SIDR).
948 1009 2
949 1010 2 BEG_OF_SIDR = .REC_ADDR;
950 1011 2 END_OF_SIDR = RM$SIDR_END();
951 1012 2
952 1013 2 ! RMS must perform an initial positioning before it begins its search for
953 1014 2 ! the next non-deleted primary data record. This positioning depends upon
954 1015 2 ! whether RMS is positioning randomly by key value, or is positioning
955 1016 2 ! sequentially. In the latter case, where RMS initially positions to will
956 1017 2 ! depend upon whether RMS had been able to position to the current SIDR
957 1018 2 ! before calling this routine. Thus, this routine is faced with four
958 1019 2 ! different ways in which it could perform this initial positioning
959 1020 2 ! depending upon the mode of access, and if the access mode is sequential,
960 1021 2 ! whether or not RMS had positioned to the current SIDR before this
961 1022 2 ! routine was called.
962 1023 2
963 1024 2 1. If RMS is positioning randomly by key value then RMS has positioned to
964 1025 2 the first SIDR whose key matches the search key according to the
965 1026 2 search characteristics, and RMS wants to begin its search for a
966 1027 2 non-deleted primary data record with the very first element in the
967 1028 2 SIDR array.
968 1029 2
969 1030 2 2. This is the first positioning attempt after a $REWIND or $CONNECT and
970 1031 2 as it is sequential there is no current SIDR. In this case RMS has
971 1032 2 positioned to the very first SIDR it could find, and begins its search
972 1033 2 for the next non-deleted primary data record with the very first
973 1034 2 element in the SIDR array.
974 1035 2
```

```
: 975 1036 2 | 3. RMS has successfully positioned to the current SIDR. In this case RMS
: 976 1037 2 | begins its search for the next non-deleted primary data record with
: 977 1038 2 | the first element that follows the current SIDR element. Note that
: 978 1039 2 | there may not be a following element in which case RMS exhausts the
: 979 1040 2 | current SIDR very quickly.
: 980 1041 2 |
: 981 1042 2 | 4. RMS was unable to position to the current SIDR before calling this
: 982 1043 2 | routine. This could only be because the entire current SIDR was
: 983 1044 2 | deleted. In this case RMS has positioned to the first SIDR with a
: 984 1045 2 | key value greater than or equal to the key of the current SIDR, and
: 985 1046 2 | it begins its search for the next non-deleted primary data record with
: 986 1047 2 | the first array element in this SIDR.
: 987 1048 2 |
: 988 1049 2 | RMSPOS_BY_COUNT (.IRAB[IRB$W_SAVE_POS]);
: 989 1050 2 |
: 990 1051 2 | If RMS is positioning randomly by key value and either the size of the
: 991 1052 2 | search key is less than the true key size or the search is for the first
: 992 1053 2 | record with an alternate key either greater-than-or-equal or greater-than
: 993 1054 2 | that of the search key, then save the search key presently located in
: 994 1055 2 | keybuffer 2 in keybuffer 5. It is necessary to save the search key because
: 995 1056 2 | if the SIDR positioned to is exhausted without locating a non-deleted
: 996 1057 2 | primary data record, it will be necessary to continue the search with the
: 997 1058 2 | next SIDR matching the search characteristics, and the search key will be
: 998 1059 2 | required in order to continue this search. In addition, it will be
: 999 1060 2 | necessary each time RMS positions to a new SIDR to extract its key into
: 1000 1061 2 | keybuffer 2, since the key of the SIDR RMS has currently positioned to
: 1001 1062 2 | might not be the same as the keys of SIDRs RMS has previously positioned
: 1002 1063 2 | to during the course of its search for a non-deleted primary data record.
: 1003 1064 2 |
: 1004 1065 2 | IF .IRAB[IRB$V_SRCHGE]
: 1005 1066 2 | OR
: 1006 1067 2 | .IRAB[IRB$V_SRCHGT]
: 1007 1068 2 | OR
: 1008 1069 2 | .IRAB[IRB$B_KEYSZ] LSSU .IDX_DFN[IDX$B_KEYSZ]
: 1009 1070 2 | THEN
: 1010 1071 2 | BEGIN
: 1011 1072 2 |   FLAGS[NEW_KEY] = 1;
: 1012 1073 2 |   CH$MOVE (.IRAB[IRB$B_KEYSZ], KEYBUF_ADDR(2), KEYBUF_ADDR(5));
: 1013 1074 2 |   END
: 1014 1075 2 |
: 1015 1076 2 | ! Regardless of how many SIDRs RMS will position to in its quest for a
: 1016 1077 2 | ! non-deleted primary data record, their key value will always be the
: 1017 1078 2 | ! same as the key in keybuffer 2 (once keybuffer 2 has a key if it doesn't
: 1018 1079 2 | ! already), and there will not be a need to extract the key out of each
: 1019 1080 2 | ! SIDR positioned to.
: 1020 1081 2 |
: 1021 1082 2 | ELSE
: 1022 1083 2 |   FLAGS[NEW_KEY] = 0;
: 1023 1084 2 |
: 1024 1085 2 | ! Continue until either the next non-deleted primary data record is
: 1025 1086 2 | ! encountered, the end-of-file is reached, or some error is encountered.
: 1026 1087 2 |
: 1027 1088 2 | NEXT :
: 1028 1089 2 | BEGIN
: 1029 1090 2 |
: 1030 1091 2 | WHILE 1
: 1031 1092 2 | DO
```

```
: 1032      1093  4      BEGIN
: 1033      1094  4
: 1034      1095  4      ! If RMS is currently positioned to the very first element of a
: 1035      1096  4      ! SDR array, and either this is first such array being scanned
: 1036      1097  4      ! during this current positioning or the search characteristics
: 1037      1098  4      ! indicate that the key of each SDR positioned to must be
: 1038      1099  4      ! extracted, then extract the key of the SDR into keybuffer 2.
: 1039      1100  4
: 1040      1101  5      IF (TESTBITSC(FLAGS[FIRST_TIME])
: 1041      1102  5          OR
: 1042      1103  5          .FLAGS[NEW_KEY])
: 1043      1104  4          AND
: 1044      1105  5          (.IRAB[IRB$W_SAVE_POS] EQLU 0)
: 1045      1106  5
: 1046      1107  4      THEN
: 1047      1108  5          BEGIN
: 1048      1109  5
: 1049      1110  5          LOCAL
: 1050      1111  5              TMP_REC_ADDR;
: 1051      1112  5
: 1052      1113  5              TMP_REC_ADDR = .REC_ADDR;
: 1053      1114  5              REC_ADDR = .BEG_OF_SDR;
: 1054      1115  5
: 1055      1116  5              REC_ADDR = .REC_ADDR + RM$REC_OVHD(-1);
: 1056      1117  5
: 1057      1118  5              IF .IDX_DFN[IDX$V_KEY_COMPR]
: 1058      1119  5                  THEN
: 1059      1120  6                      BEGIN
: 1060      1121  6
: 1061      1122  6                          GLOBAL REGISTER
: 1062      1123  6                              R_BDB;
: 1063      1124  6
: 1064      1125  6                              AP = 1;
: 1065      1126  6                              RM$RECORD_KEY (KEYBUF_ADDR(2));
: 1066      1127  6                              END
: 1067      1128  5              ELSE
: 1068      1129  5                  CH$MOVE (.IDX_DFN[IDX$B_KEYSZ], .REC_ADDR, KEYBUF_ADDR(2));
: 1069      1130  5
: 1070      1131  5              REC_ADDR = .TMP_REC_ADDR;
: 1071      1132  4              END;
: 1072      1133  4
: 1073      1134  4      ! Continue searching for the next non-deleted primary data record
: 1074      1135  4      ! until either it is found, an error occurs, the end of the SDR
: 1075      1136  4      ! is encountered, or no SDR with a key matching the search key
: 1076      1137  4      ! according to the search characteristics remains to be scanned for an
: 1077      1138  4      ! array element pointing to a non-deleted primary data record.
: 1078      1139  4
: 1079      1140  4      WHILE .REC_ADDR LSSA .END_OF_SDR
: 1080      1141  4      DO
: 1081      1142  5          BEGIN
: 1082      1143  5
: 1083      1144  5              ! Starting with the current SDR array element, search for the
: 1084      1145  5              ! first element in the SDR array that is not marked deleted
: 1085      1146  5              ! until either one is found or the end of the SDR is encountered.
: 1086      1147  5
: 1087      1148  7              WHILE ((.REC_ADDR LSSA .END_OF_SDR)
: 1088      1149  6                  AND
```

```

: 1089      1150 6
: 1090      1151 5
: 1091      1152 6
: 1092      1153 6
: 1093      1154 6
: 1094      1155 6
: 1095      1156 5
: 1096      1157 5
: 1097      1158 5
: 1098      1159 5
: 1099      1160 5
: 1100      1161 5
: 1101      1162 5
: 1102      1163 5
: 1103      1164 5
: 1104      1165 5
: 1105      1166 6
: 1106      1167 6
: 1107      1168 6
: 1108      1169 6
: 1109      1170 6
: 1110      1171 6
: 1111      1172 6
: 1112      1173 6
: 1113      1174 6
: 1114      1175 6
: 1115      1176 6
: 1116      1177 6
: 1117      1178 6
: 1118      1179 6
: 1119      1180 6
: 1120      1181 6
: 1121      1182 6
: 1122      1183 6
: 1123      1184 6
: 1124      1185 6
: 1125      1186 6
: 1126      1187 6
: 1127      1188 6
: 1128      1189 6
: 1129      1190 7
: 1130      1191 6
: 1131      1192 6
: 1132      1193 6
: 1133      1194 6
: 1134      1195 6
: 1135      1196 6
: 1136      1197 6
: 1137      1198 6
: 1138      1199 6
: 1139      1200 6
: 1140      1201 6
: 1141      1202 6
: 1142      1203 6
: 1143      1204 7
: 1144      1205 7
: 1145      1206 7

      .REC_ADDR[IRC$V_DELETED])
DO
  BEGIN
    RMSGETNXT_ARRAY();
    IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
  END;

  ! If RMS has not run off the end of the current SIDR array it is
  ! searching, but has successfully positioned to an element in the
  ! SIDR array that is not marked deleted, then attempt to access the
  ! primary data record that it points to, and determine whether or
  ! not it is marked deleted.
  IF .REC_ADDR LSSA .END_OF_SIDR
  THEN
    BEGIN
      LOCAL
        RFA_ID,
        RFA_VBN;

      ! Attempt to access the primary data record while holding on to
      ! the SIDR bucket lock; waiting for the lock on the primary data
      ! bucket to be released if someone else has it while holding
      ! onto the SIDR bucket lock if necessary.
      RM$EXT_ARRY_RFA (RFA_VBN, RFA_ID);
      STATUS = RMS$FOLLOW_PTR (.RFA_VBN, .RFA_ID);

      ! If RMS has successfully found the next non-deleted primary
      ! data record, or some serious error has been encountered,
      ! then terminate the search for the next non-deleted primary
      ! data record. If the error encountered was just that the
      ! primary data record accessed had been deleted, then RMS will
      ! be able to continue the search for the next non-deleted
      ! primary data record with the next element in the SIDR array
      ! currently being searched which is not marked deleted.
      IF .STATUS<0,16> NEQU RMSERR(DEL)
      THEN
        LEAVE NEXT;

      ! If the primary data record positioned to is deleted, but the
      ! SIDR array element is not marked as such, then alleviate this
      ! discrepancy provided the user has accessed the file for
      ! writing, before continuing on with the search for the next
      ! non-deleted primary data record. If the current SIDR array
      ! element is marked RU_DELETE, and RMS is to RU_DELETE it, then
      ! in this case there is no need to do anything.
      IF .IFAB[IFB$V_WRTACC]
      AND
      NOT (.REC_ADDR[IRC$V_RU_DELETE]
      AND
      .IRAB[IRB$V_RU_DELETE])
```

```
: 1146      1207  6
: 1147      1208  7
: 1148      1209  7
: 1149      1210  7
: 1150      1211  7
: 1151      1212  7
: 1152      1213  7
: 1153      1214  7
: 1154      1215  7
: 1155      1216  7
: 1156      1217  7
: 1157      1218  7
: 1158      1219  7
: 1159      1220  7
: 1160      1221  7
: 1161      1222  7
: 1162      1223  7
: 1163      1224  7
: 1164      1225  7
: 1165      1226  7
: 1166      1227  7
: 1167      1228  7
: 1168      1229  7
: 1169      1230  7
: 1170      1231  7
: 1171      1232  7
: 1172      1233  7
: 1173      1234  7
: 1174      1235  7
: 1175      1236  7
: 1176      1237  7
: 1177      1238  8
: 1178      1239  8
: 1179      1240  8
: 1180      1241  8
: 1181      1242  8
: 1182      1243  8
: 1183      1244  8
: 1184      1245  8
: 1185      1246  8
: 1186      1247  8
: 1187      1248  8
: 1188      1249  8
: 1189      1250  8
: 1190      1251  7
: 1191      1252  7
: 1192      1253  7
: 1193      1254  7
: 1194      1255  7
: 1195      1256  7
: 1196      1257  7
: 1197      1258  7
: 1198      1259  7
: 1199      1260  7
: 1200      1261  8
: 1201      1262  8
: 1202      1263  8

      THEN
      BEGIN
      GLOBAL REGISTER
      COMMON_IO_STR;

      LOCAL
      SAVE_REC_ADDR;

      ! Retrieve the address of the SIDR bucket and mark it dirty.
      BDB = .IRAB[IRB$L_CURBDB];
      BDB[BDB$V_DRT] = T;
      BKT_ADDR = .BDB[BDB$L_ADDR];

      ! Save the address of the current SIDR array element before
      ! attempting to delete any space.
      SAVE_REC_ADDR = .REC_ADDR;
      REC_ADDR[IRB$V_RU_DELETE] = 0;

      ! If RMS is able to only delete the space occupied by the
      ! current SIDR array element's RRV pointer, then RMS may
      ! continue its search for a non-deleted primary data record
      ! within the current SIDR array after re-determining the
      ! address of the end of the current SIDR array.
      IF RMSSQUISH_SIDR (1, .BEG_OF_SIDR)
      THEN
      IF .REC_ADDR EQLU .SAVE_REC_ADDR
      THEN
      BEGIN
      REC_ADDR = .BEG_OF_SIDR;
      END_OF_SIDR = RMSSIDR-END();
      REC_ADDR = .SAVE_REC_ADDR;
      END

      ! If RMS is able to delete the entire current SIDR
      ! because all the elements contained within it have
      ! been deleted, then leave the loop responsible for
      ! searching the current SIDR so that conditions might
      ! be set up to attempt to search the next SIDR array
      ! in the alternate index.
      ELSE
      EXITLOOP

      ! If RMS is able to only mark the current SIDR array as
      ! deleted without recovering any space then make sure the
      ! state bit IRB$V_RU_DELETE is clear (if it was set) and
      ! continue the search for a non-deleted primary data record
      ! with the next SIDR array element.
      ELSE
      BEGIN
      IRAB[IRB$V_RU_DELETE] = 0;
      RMSGETNXT_ARRAY();
```

```

: 1203      1264  8      IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
: 1204      1265  7      END;
: 1205      1266  7      END
: 1206      1267  7
: 1207      1268  7      ! If the file has not been opened for write access or there are
: 1208      1269  7      ! really no changes that need to be made then the SIDR element
: 1209      1270  7      ! can not be marked deleted, and RMS must immediately position
: 1210      1271  7      ! to the next SIDR array element. This is because having
: 1211      1272  7      ! unexpectantly encountered a deleted primary data record RMS
: 1212      1273  7      ! will want to continue its search for the next non-deleted
: 1213      1274  7      ! SIDR element. Since RMS always starts such a search with the
: 1214      1275  7      ! current element, and since RMS was unable to mark the current
: 1215      1276  7      ! SIDR array element deleted, RMS would end up positioning to
: 1216      1277  7      ! the current element as if it was the next element.
: 1217      1278  7      ! Immediately positioning to the next element prevents this.
: 1218      1279  7
: 1219      1280  6      ELSE
: 1220      1281  7      BEGIN
: 1221      1282  7      RMSGETNXT ARRAY();
: 1222      1283  7      IRAB[IRB$W_SAVE_POS] = .IRAB[IRB$W_SAVE_POS] + 1;
: 1223      1284  7      IRAB[IRB$V_RU_DELETE] = 0;
: 1224      1285  6      END;
: 1225      1286  5      END;
: 1226      1287  5
: 1227      1288  4      END;
: 1228      1289  4
: 1229      1290  4      ! Having exhausted this current SIDR array in the search for the next
: 1230      1291  4      ! non-deleted primary data record, RMS positions to the next SIDR
: 1231      1292  4      ! in order to continue the search. NOTE that if the search
: 1232      1293  4      ! characteristics require the original search key, it is restored to
: 1233      1294  4      ! keybuffer 2 from keybuffer 5 before initiating the search. This
: 1234      1295  4      ! will always be the case when the key of the next SIDR RMS positions
: 1235      1296  4      ! to maybe different from the key of the current SIDR.
: 1236      1297  4
: 1237      1298  4      IF .FLAGS[NEW_KEY]
: 1238      1299  4      THEN
: 1239      1300  4      CH$MOVE (.IRAB[IRB$B_KEYSZ], KEYBUF_ADDR(5), KEYBUF_ADDR(2));
: 1240      1301  4
: 1241      1302  4      IRAB[IRB$V_NORLS_RNF] = 1;
: 1242      1303  4      RETURN_ON_ERROR (RMS$CSEARCH_TREE(), IRAB[IRB$V_PRM] = 0);
: 1243      1304  4
: 1244      1305  4      ! Reset all the fields necessary to search a SIDR array starting
: 1245      1306  4      ! from its first element.
: 1246      1307  4
: 1247      1308  4      BEG_OF_SIDR = .REC_ADDR;
: 1248      1309  4      END_OF_SIDR = RMS$SIDR_END();
: 1249      1310  4
: 1250      1311  4      REC_ADDR = RMS$SIDR_FIRST(0);
: 1251      1312  4      IRAB[IRB$W_SAVE_POS] = 0;
: 1252      1313  3      END;
: 1253      1314  3
: 1254      1315  3
: 1255      1316  2      END;
: 1256      1317  2      ! {of NEXT}
: 1257      1318  2
: 1258      1319  2      ! If some serious error was encountered in RMS's search for the next
: 1259      1320  2      ! non-deleted primary data record such that the search could no longer
:      ! be continued, then RMS releases any bucket locks the process has
```

```

: 1260      1321 2      ! outstanding (primary or SIDR but not both), and returns the error.
: 1261      1322 2
: 1262      1323 2      ! There is one error case which is handled differently. A record not found
: 1263      1324 2      ! error on a search for a primary data record is a serious error. To signal
: 1264      1325 2      ! to a caller of this routine that in this particular case this routine
: 1265      1326 2      ! should never be recalled, RMS returns an invalid internal condition or
: 1266      1327 2      ! bug error after storing the RNF status in the RAB's RAB$$_STV field.
: 1267      1328 2
: 1268      1329 2      IF NOT .STATUS
: 1269      1330 2      THEN
: 1270      1331 2          BEGIN
: 1271      1332 2
: 1272      1333 2          GLOBAL REGISTER
: 1273      1334 2              COMMON_IO_STR;
: 1274      1335 2
: 1275      1336 2          IF .IRAB[IRB$_CURBDB] NEQU 0
: 1276      1337 2          THEN
: 1277      1338 2              RELEASE (IRAB[IRB$_CURBDB]);
: 1278      1339 2
: 1279      1340 2          IF .STATUS<0,16> EQLU RMSERR(RNF)
: 1280      1341 2          THEN
: 1281      1342 2              BEGIN
: 1282      1343 2                  RAB[RAB$_STV] = .STATUS;
: 1283      1344 2                  STATUS = RMSERR(BUG);
: 1284      1345 2                  END;
: 1285      1346 2
: 1286      1347 2              IRAB[IRB$_PRM] = 0;
: 1287      1348 2              RETURN .STATUS;
: 1288      1349 2              END
: 1289      1350 2
: 1290      1351 2      ! If RMS has been successful in positioning to the next non-deleted
: 1291      1352 2      ! primary data record, then save all information necessary to position
: 1292      1353 2      ! to this SIDR array element as if it was the current element (because it
: 1293      1354 2      ! soon will be), release the SIDR bucket, and return success.
: 1294      1355 2
: 1295      1356 2      ELSE
: 1296      1357 2          BEGIN
: 1297      1358 2
: 1298      1359 2          GLOBAL REGISTER
: 1299      1360 2              COMMON_IO_STR;
: 1300      1361 2
: 1301      1362 2          LOCAL
: 1302      1363 2              SAVE_REC_ADDR;
: 1303      1364 2
: 1304      1365 2          BDB = .IRAB[IRB$_NXTBDB];
: 1305      1366 2          IRAB[IRB$_NXTBDB] = 0;
: 1306      1367 2
: 1307      1368 2          IRAB[IRB$_RFA_VBN] = .BDB[BDB$_VBN];
: 1308      1369 2
: 1309      1370 2          SAVE_REC_ADDR = .REC_ADDR;
: 1310      1371 2          REC_ADDR = .BEG OF SIDR;
: 1311      1372 2          RMSSIDR FIRST (T; IRAB[IRB$_FIRST_VBN], IRAB[IRB$_FIRST_ID]);
: 1312      1373 2          REC_ADDR = .SAVE_REC_ADDR;
: 1313      1374 2
: 1314      1375 2
: 1315      1376 2          IF .IRAB[IRB$_PRM]
: 1316      1377 2          THEN
```

```
: 1317      1378 3      BDB[BDB$V_PRM] = 1;
: 1318      1379 3
: 1319      1380 3      RM$RLSBKT(0);
: 1320      1381 3      IRAB[IRB$V_PRM] = 0;
: 1321      1382 3
: 1322      1383 3      RETURN RMSSUC();
: 1323      1384 3      END;
: 1324      1385 2
: 1325      1386 1      END;
```

! (of the routine RM\$SEARCH_SIDR)

```
3C BB 00000 RM$SEARCH_SIDR::
      5E      1C C2 00002      POSHR #^M<R2,R3,R4,R5>      0866
08 AE      01 88 00005      SUBL2 #28, SP      1005
04 AE      56 D0 00009      BISB2 #1, FLAGS      1010
      0000G 30 0000D      MOVL REC_ADDR, BEG_OF_SIDR      1011
      50 D0 00010      MOVL R0, END_OF_SIDR
      51 76 A9 3C 00013      MOVZWL 118(IRAB), R1      1049
      B3 10 00017      BSBB RM$POS BY COUNT
0D 42 A9      04 E0 00019      BBS #4, 66(IRAB), 1$      1065
08 42 A9      01 E0 0001E      BBS #1, 66(IRAB), 1$      1067
20 A7      00A6 C9 91 00023      CMPB 166(IRAB), 32(IDX_DFN)      1069
      1A 1E 00029      BGEQU 2$
08 AE      02 88 0002B 1$:      BISB2 #2, FLAGS      1072
      51 00A6 C9 9A 0002F      MOVZBL 166(IRAB), R1      1073
      50 00B4 CA 3C 00034      MOVZWL 180(IRAB), R0
      60 B940 DF 00039      PUSHAL @96(IRAB)[R0]
9E 60 B940      51 28 0003D      MOVC3 R1, @96(IRAB)[R0], @ (SP)+
      04 11 00043      BRB 3$      1065
08 AE      02 8A 00045 2$:      BICB2 #2, FLAGS      1083
05 08 AE      00 E4 00049 3$:      BBSC #0, FLAGS, 4$      1101
44 08 AE      01 E1 0004E      BBC #1, FLAGS, 7$      1103
      76 A9 B5 00050 4$:      TSTW 118(IRAB)      1105
      3F 12 00056      BNEQ 7$
      56 D0 00058      MOVL REC_ADDR, TMP_REC_ADDR      1113
      04 AE D0 0005C      MOVL BEG_OF_SIDR, REC_ADDR      1114
      01 CE 00060      MNEGL #1, R1      1116
      0000G 30 00063      BSBW RM$REC OVHD
      50 C0 00066      ADDL2 R0, REC_ADDR
14 1C A7      06 E1 00069      BBC #6, 28(IDX_DFN), 5$      1118
      5C 01 D0 0006E      MOVL #1, AP      1125
      50 00B4 CA 3C 00071      MOVZWL 180(IRAB), R0      1126
      60 B940 9F 00076      PUSHAB @96(IRAB)[R0]
      0000G 30 0007A      BSBW RM$RECORD_KEY
      5E 04 C0 0007D      ADDL2 #4, SP
      11 11 00080      BRB 6$      1118
      51 20 A7 9A 00082 5$:      MOVZBL 32(IDX_DFN), R1      1129
      50 00B4 CA 3C 00086      MOVZWL 180(IRAB), R0
      50 60 A9 C0 0008B      ADDL2 96(IRAB), R0
60 66 51 28 0008F      MOVC3 R1, (REC_ADDR), (R0)
      5C 0C AE D0 00093 6$:      MOVL TMP_REC_ADDR, REC_ADDR      1131
      5E 56 D1 00097 7$:      CMPL REC_ADDR, END_OF_SIDR      1140
      03 1F 0009A      BLSSU 8$
```

			0094	31	0009C	BRW	15\$		
	6E		56	D1	0009F	CMPL	REC_ADDR, END_OF_SIDR		1148
			F3	1E	000A2	BGEQU	7\$		
08		66	02	E1	000A4	BBC	#2, (REC_ADDR), 9\$		1150
			0000G	30	000A8	BSBW	RM\$GETNXT_ARRAY		1154
		76	A9	B6	000AB	INCW	118(IRAB)		1155
			EF	11	000AE	BRB	8\$		1148
			E5	1E	000B0	BGEQU	7\$		1164
		14	AE	9F	000B2	PUSHAB	RFA_ID		1177
		1C	AE	9F	000B5	PUSHAB	RFA_VBN		
			0000G	30	000B8	BSBW	RM\$EXT_ARRAY_RFA		
	5E		08	C0	000BB	ADDL2	#8, SP		
		14	AE	DD	000BE	PUSHL	RFA_ID		1179
		1C	AE	DD	000C1	PUSHL	RFA_VBN		
			FDD9	30	000C4	BSBW	RM\$FOLLOW_PTR		
	5E		08	C0	000C7	ADDL2	#8, SP		
10	AE		50	D0	000CA	MOVL	R0, STATUS		
8262	8F	10	AE	B1	000CE	CMPW	STATUS, #33378		1190
			03	13	000D4	BEQL	10\$		
			009F	31	000D6	BRW	18\$		
	49	06	AA	E9	000D9	BLBC	6(IFAB), 13\$		1202
	66		05	E1	000DD	BBC	#5, (REC_ADDR), 11\$		1204
05			05	E0	000E1	BBS	#5, 7(IRAB), 13\$		1206
40	07		A9	D0	000E6	MOVL	32(IRAB), BDB		1218
		20	A9	D0	000EA	BISB2	#2, 10(BDB)		1219
	0A		A4	D0	000EE	MOVL	24(BDB), BKT_ADDR		1220
		18	A4	D0	000F2	MOVL	REC_ADDR, SAVE_REC_ADDR		1225
			56	D0	000F5	BICB2	#32, (REC_ADDR)		1226
			20	8A	000F8	PUSHL	BEG_OF_SIDR		1234
		04	AE	DD	000FB	PUSHL	#1		
			01	DD	000FD	BSBW	RM\$SQUISH_SIDR		
			0000G	30	00100	ADDL2	#8, SP		
	5E		08	C0	00103	BLBC	R0, 12\$		
	14		50	E9	00106	CMPL	REC_ADDR, SAVE_REC_ADDR		1236
	52		56	D1	00109	BNEQ	15\$		
			28	12	0010B	MOVL	BEG OF SIDR, REC_ADDR		1239
	56	04	AE	D0	0010F	BSBW	RM\$SIDR_END		1240
			0000G	30	00112	MOVL	R0, END_OF_SIDR		
	6E		50	D0	00115	MOVL	SAVE_REC_ADDR, REC_ADDR		1241
	56		52	D0	00118	BRB	14\$		1236
			16	11	0011A	BICB2	#32, 7(IRAB)		1262
	07	A9	20	8A	0011E	BSBW	RM\$GETNXT_ARRAY		1263
			0000G	30	00121	INCW	118(IRAB)		1264
		76	A9	B6	00124	BRB	14\$		1202
			0A	11	00126	BSBW	RM\$GETNXT_ARRAY		1282
			0000G	30	00129	INCW	118(IRAB)		1283
		76	A9	B6	0012C	BICB2	#32, 7(IRAB)		1284
	07	A9	20	8A	00130	BRW	7\$		1164
			FF64	31	00133	BBC	#1, FLAGS, 16\$		1298
14	08	AE	01	E1	00138	MOVZBL	166(IRAB), R1		1300
		51	00A6	C9	0013D	MOVZWL	180(IFAB), R0		
		50	00B4	CA	00142	PUSHAL	@96(IRAB)[R0]		
			60	B940	00146	MOVC3	R1, @ (SP)+, @96(IRAB)[R0]		
			51	28	0014C	BISB2	#32, 66(IRAB)		1302
	42	A9	20	88	00150	BSBW	RM\$CSEARCH_TREE		1303
			0000G	30	00153	BLBS	STATUS, 17\$		
			50	E8	00156	BICB2	#128, 66(IRAB)		
	07		80	8F					
	42	A9							

04	AE	4F	11	0015B	BRB	21\$	:	
		56	D0	0015D	17\$:	MOVL	REC_ADDR, BEG_OF_SIDR	1308
		0000G	30	00161	BSBW	RMSSIDR_END	:	1309
	6E	50	D0	00164	MOVL	R0, END_OF_SIDR	:	
		7E	D4	00167	CLRL	-(SP)	:	1311
		0000G	30	00169	BSBW	RMSSIDR_FIRST	:	
	5E	04	C0	0016C	ADDL2	#4, SP	:	
	56	50	D0	0016F	MOVL	R0, REC_ADDR	:	
		76	A9	B4	00172	CLRW	118(IRAB)	1312
		FED1	31	00175	BRW	3\$	:	1091
	32	10	AE	E8	00178	18\$:	BLBS	STATUS, 22\$
		20	A9	D5	0017C	TSTL	32(IRAB)	1329
			0F	13	0017F	BEQL	19\$	1336
	54	20	A9	D0	00181	MOVL	32(IRAB), BDB	1338
		20	A9	D4	00185	CLRL	32(IRAB)	
			7E	D4	00188	CLRL	-(SP)	
		0000G	30	0018A	BSBW	RMSRLSBKT	:	
	5E	04	C0	0018D	ADDL2	#4, SP	:	
82B2	8F	10	AE	B1	00190	19\$:	CMPW	STATUS, #33458
			0B	12	00196	BNEQ	20\$	1340
0C	A8	10	AE	D0	00198	MOVL	STATUS, 12(RAB)	1343
10	AE	8434	8F	3C	0019D	MOVZWL	#33844, STATUS	1344
42	A9	80	8F	8A	001A3	20\$:	BICB2	#128, 66(IRAB)
	50	10	AE	D0	001A8	MOVL	STATUS, R0	1357
			40	11	001AC	21\$:	BRB	24\$
	54	3C	A9	D0	001AE	22\$:	MOVL	60(IRAB), BDB
		3C	A9	D4	001B2	CLRL	60(IRAB)	1365
70	A9	1C	A4	D0	001B5	MOVL	28(BDB), 112(IRAB)	1366
	53		56	D0	001BA	MOVL	REC_ADDR, SAVE_REC_ADDR	1368
	56	04	AE	D0	001BD	MOVL	BEG_OF_SIDR, REC_ADDR	1370
			01	DD	001C1	PUSHL	#1	1371
		0000G	30	001C3	BSBW	RMSSIDR_FIRST	:	1372
	5E	04	C0	001C6	ADDL2	#4, SP	:	
7C	A9		51	D0	001C9	MOVL	R1, 124(IRAB)	
0082	C9		52	B0	001CD	MOVW	R2, 130(IRAB)	
	56		53	D0	001D2	MOVL	SAVE_REC_ADDR, REC_ADDR	1373
		42	A9	95	001D5	TSTB	66(IRAB)	1376
			04	18	001D8	BGEQ	23\$	
0A	A4		08	88	001DA	BISB2	#8, 10(BDB)	1378
			7E	D4	001DE	23\$:	CLRL	-(SP)
		0000G	30	001E0	BSBW	RMSRLSBKT	:	1380
	5E	04	C0	001E3	ADDL2	#4, SP	:	
42	A9	80	8F	8A	001E6	BICB2	#128, 66(IRAB)	1381
	50		01	D0	001EB	MOVL	#1, R0	1383
	5E		1C	C0	001EE	24\$:	ADDL2	#28, SP
			3C	BA	001F1	POPR	#^M<R2,R3,R4,R5>	1386
			05	001F3	RSB		:	

; Routine Size: 500 bytes, Routine Base: RMSRMS3 + 0160

: 1326	1387	1
: 1327	1388	1 END
: 1328	1389	1
: 1329	1390	0 ELUDOM

RM3SSIDR
V04-000

RM\$SEARCH_SIDR

H 11
16-Sep-1984 02:07:52
14-Sep-1984 13:01:41

VAX-11 Bliss-32 V4.0-742
DISK\$VMSMASTER:[RMS.SRC]RM3SSIDR.B32;1
Page 31
(4)

PSECT SUMMARY

Name	Bytes	Attributes
RM\$RMS3	852	NOVEC,NOWRT, RD , EXE,NOSHR, GBL, REL, CON, PIC,ALIGN(2)

Library Statistics

File	----- Total	Symbols Loaded	----- Percent	Pages Mapped	Processing Time
_\$255\$DUA28:[RMS.OBJ]RMS.L32;1	3109	70	2	154	00:00.4

COMMAND QUALIFIERS

BLISS/CHECK=(FIELD,INITIAL,OPTIMIZE)/LIS=LIS\$:RM3SSIDR/OBJ=OBJ\$:RM3SSIDR MSRC\$:RM3SSIDR/UPDATE=(ENH\$:RM3SSIDR)

: Size: 852 code + 0 data bytes
: Run Time: 00:23.7
: Elapsed Time: 00:48.1
: Lines/CPU Min: 3513
: Lexemes/CPU-Min: 14567
: Memory Used: 195 pages
: Compilation Complete

0328

AH-BT13A-SE
VAX/VMS V4.0

DIGITAL EQUIPMENT CORPORATION
CONFIDENTIAL AND PROPRIETARY